# MAGATARI™

VOLUME 1 NUMBER 7

## for the ATARI 400/800

1 REVERSI

2 THE DATABOSS #1

3 PROGRAMMER'S KEYBOARD
our utility of the month

4 6502 TUTORIAL #1

5 MUSIC AND SOUND EFFECTS

$12.95 Cassette, $17.95 Disk


**programmer's institute**

a division of **FUTUREHOUSE™**

310½ West Franklin Street

P.O. Box 3470, Chapel Hill, NC 27514

**1-800-334-SOFT**

***orderline***

## Editor's Notes — Andrew A. Hock

We are pleased to welcome Chris Aiken to ***Magatari*** as a regular columnist and program contributor. Chris is an excellent game and utility programmer, and has recently completed writing **SKY WARRIOR!™**, an arcade-style game with superb graphics. We're sure that our subscribers will look forward to Chris's input to ***Magatari.***

Beginning with this issue, we are publishing a database management system. The 'shell' for creating files and editing records is included in this month's issue, and additional features such as sorting, merging files, printer options, calculator style features, and more in future issues. For those of you who don't understand what a database is, ex- in these pages for part 1 of **THE DATABOSS™**.

Other programs in our current issue include REVERSI board game, our utility of the month - PROGRAMMER'S KEYBOARD, some excellent sound effects and songs by Dana DeVice, and the first in our series of 6502 Machine Language tutorials.

We hope everyone is pleased with the new look of ***Magatari.*** We have undergone many changes in the past few months and will go through many more. All of us here at ***Magatari*** hope you, our subscribers, enjoy the changes and additions, and will continue to make ***Magatari*** the Number 1 'magnetic' magazine.


## STAFF OF MAGATARI

FUTUREHOUSE, *Publisher*
Andrew A. Hock, *Editor*
Chris F. Stewart, *Managing Editor*
James A. Douglas, *Marketing Director*
Debbie Hirsch, *Editorial Assistant*

*Associate Editors:*
Darren DeLoach,
c/o MAGATARI, P.O. Box 3191,
Chapel Hill, NC 27514
Chris Aiken
c/o MAGATARI, P.O. Box 3191,
Chapel Hill, NC 27514

## Loading Instructions

Here are the programs included in Volume 1 Number 7. For all cassette subscribers, the line counter numbers are listed to the right of the names of the programs:

| PROGRAM NAMES | COUNTER# |
|---|---|
| THE DATABOSS | 10 |
| UTILITY-KEYBOARD | 101 |
| REVERSI | 114 |
| MACHINE LANGUAGE | 178 |
| SOUND EFFECTS | 238 |
| SOUND DEMO | 301 |

Cassette users should advance their tape recorder to the specified line number, type in CLOAD, and press [RETURN] twice (the programs are saved on both sides of the tape).

All diskette subscribers should insert the ***Magatari*** diskette into their drive, and turn on their machine. If you wish to load the programs separately, do the following:

| PROGRAM NAMES | INSTRUCTION |
|---|---|
| MENU PROGRAM | RUN"D:MAGATARI" |
| DATABOSS | RUN"D:DATABOSS" |
| REVERSI | RUN:"D:REVERSI" |
| MACHINE LANG. | RUN"D:MACHINE" |
| SOUND EFFECTS | RUN"D:SOUNDEF" |
| SOUND DEMO. | RUN"D:SOUNDEMO" |

## The Programs

## THE DATABOSS™ - part 1

A database is a program (or set of programs) that allows you to create any file desired and manipulate the files (i.e. merge and sort files, print reports complete with totals, format entries, do calculations on fields, etc.). For example, you can create a mailing list file, an accounts receivable file complete with 30, 60 and 90 day maintenance, a home inventory file, a payroll file, stock file . . . just about any file you want.

Part 1 of our database let's you create a file, print the file to the screen, and save it on cassette or disk. The disk version uses the random access method of accessing disk files, while the cassette version is sequential and the length of a file depends on the amount of RAM your ATARI has left after loading THE DATABOSS™.

### DISK VERSION

The main menu of THE DATABOSS™ looks like this:

```
THE DATABOSS™ [VER 1.0]
copyright © 1982 by FUTUREHOUSE

[A] Create file          [B] Load file
[C] Select print         [D] File inquiry
[E] Edit sort links      [F] File merge
[G] Format disk          [H] Disk directory
[I] Edit file spec       [J] End this job

What is your choice?
```

Options C, E and I will be implemented in a future issue. Creating a file and loading a file are the same as last issue. For the benefit of those who did not receive the last issue, we have included the documentation for these options in this issue. Upon selecting Option A (Create file), the following screen will appear:

```
THE DATABOSS™ - - CREATE MODE

press [M]enu or [RETURN] to continue
```

In our example, we will create a home inventory file. The file will contain records, each with the following fields and field characteristics:

| TITLE | LENGTH | TYPE |
|---|---|---|
| 1. NAME OF ITEM | 25 char | alphanumeric |
| 2. CONDITION | 2 char | alphanumeric |
| 3. QUANTITY | 3 char | numeric |
| 4. PRICE PAID | 10 char | dollars/cents |
| 5. PRESENT VALUE | 10 char | dollars/cents |

After pressing the [RETURN] key, the next screen will appear:

THE DATABOSS™ - - CREATE MODE

☐ number of fields per record

In our home inventory example there are 5 fields (you can have up to 99 fields), so press 5 then [RETURN].

The next screen appears:

THE DATABOSS™ - - CREATE MODE

enter name of this application

The name of the application can be up to 25 characters long. You can call it anything you wish, but for the sake of clarity, we'll call it 'HOME INVENTORY FILE.' Type this in, then press [RETURN].

THE DATABOSS™ - - CREATE MODE

enter name of this application

HOME INVENTORY FILE

enter filename for application
DISK FORMAT WILL BE "D:filename.DAT

The disk file name can be up to 8 characters long. We'll type in 'HOMEINVT'.

The keyboard will buzz, warning you that you are about to enter the create mode, and you will have two options. Key in [M] to return to the main menu (i.e. if you accidentally hit the wrong key), or hit [RETURN] to continue to the first step of creating a file.

The next screen looks like this:

THE DATABOSS™ - - CREATE MODE

enter title for this field

The name of the first field in our example is 'NAME OF ITEM'. Type this in, the press [RETURN].

THE DATABOSS™ - - CREATE MODE

enter title for this field

NAME OF ITEM

enter length of this field (-1-99)

☐

In our example, the length of this field was selected to be 25 characters, so key in 25.

THE DATABOSS™ - - CREATE MODE

enter title of this field

NAME OF ITEM

enter length of this field (01-99)

25

SELECT FIELD EDIT FORMAT:

1 alphanumeric 2 numeric
3 date mm/dd/yy 4 dollars/cents

Since the NAME OF ITEM field should be able to hold any character (numbers or letters), select Option 1 (alphanumeric characters).

You will then be given the following prompt:

ARE THESE ENTRIES CORRECT [Y/N]?

If you've made any errors, simply press 'N' and repeat the above process. If the entries are correct, press 'Y'.

Repeat the above process for the next four fields, using the features described for each field. After completing the process, the following screen will appear:

THE DATABOSS™ - - CREATE MODE

ENTER # OF RECORDS IN FILE

Depending on the size of the record, you can have up to 999 records in a file. For our purposes, we'll create 200 records for our HOME INVENTORY FILE. Type in 200.

You will then be asked to:

INSERT DISKETTE IN DRIVE 1

Insert a formatted diskette in your drive (you can also format a diskette by selecting Option 5 from the main menu). Then press any key to start the create file process. The busy light on your drive will turn on, and you will be informed the program is NOW CREATING DATABASE FILE. After the process is complete, control will return to the main menu.

Select 1, since we'll want to keep this field sorted. The next prompt will appear. There is also an option now for a disk directory. Simply insert the disk you wish to see the directory for, and select Option H.

After loading or creating a file, select Option D to edit the file. In our example of a Home Inventory File, the following screen will appear:

THE DATABOSS™ - FILE INQUIRY MODE

REC # FILE: HOME INVENTORY FILE

NAME OF ITEM

CONDITION QUANTITY

PRICE PAID PRESENT VALUE

[E]dit [M]enu [N]ext [L]ast [S]earch
[F]orward [B]ack [D]elete [R]ecord #

In case you didn't notice, THE DATABOSS™ automatically formats the screen for you, centering all fields, and printing them in inverse. You can also see the exact length assigned every field, because the screens are all in inverse. To add a record, select [E] for EDIT. There is not an option to [A]dd or [C]hange. If you want to add a record, access the record you wish to add by using [N]ext or [L]ast to advance or go back a record at a time; or select [R] for record #, and enter the record number desired to access the record immediately. Then select [E] to enter the data.

If you make a mistake or wish to change the data in a field, simply press [E]. If you do not want to change the field that the cursor is over, press [RETURN] without entering any data and the cursor will advance to the next field, without changing the data in the previous field.

If you wish to search the file for a particular record, select [S]earch. You will be asked the field number of the field you want

to make a search, and then the data to search for. The search will begin, and stop at the first record that meets the criterion of the search. If you wish to continue the search, select [S]earch again and key [C] for continue to the field number prompt. When finished editing, select [M] to return to the main menu.

**CASSETTE VERSION**

The cassette version of THE DATABOSS™ accesses record sequentially, and the number of records you can have in any file you create is dependent on the amount of RAM available as opposed to the amount of disk space (as in the disk version). The main menu of the cassette version looks like this:

---

THE DATABOSS™ [VER 1.0]
**copyright © 1982 by FUTUREHOUSE**

---

**1** create new file
**2** load existing file
**3** print file
**4** file inquiry
**5** save file
**6** end this job

---

What is your choice?

---

The only difference is the save file option (Option 5). In the tape version, you must save a file after creating and entering data . . . it is not done automatically as in the disk version. Also, selecting Option 6 will end the program, since there is no ***Magatari*** menu to return to.

## REVERSI

Here's our version of **REVERSI. REVERSI** is a board game for two players. The object of the game is to have more pieces on the board at the end of the game than your opponent. Each player starts the game with two pieces on the board, and each adds a piece to the board in turn. There are only two rules:

1) A move is legal only if the addition of a piece turns over at least one of the opponent's pieces.

2) Piece(s) are turned over when one player's pieces are situated at both ends of the opposing player's pieces.

Here is a one-player game, and a two-player game. The one-player game pits you against the computer, while the two-player game allows you to play another human. When playing either game, player(s) have 4 options:

1) they can make a move by moving the blinking cursor with joysticks, over the space where they wish to place a piece, and press the trigger button.

2) they can let the computer make a move for them by pressing the 'C' key.

3) they can save the game by pressing the 'S' key, and following the prompts

4) they can load a previous game saved, by pressing the 'L' key, and following the prompts.

NOTE: Sometime it takes the computer up to 3 minutes to make a move . . . so please be patient!

## MUSIC AND SOUND EFFECTS

**. . . on your ATARI 400/800.**

Here we have a library of sound effects that you can use with your programs. Included are explosions, and other effects like wind, walking, machine gun fire, etc.

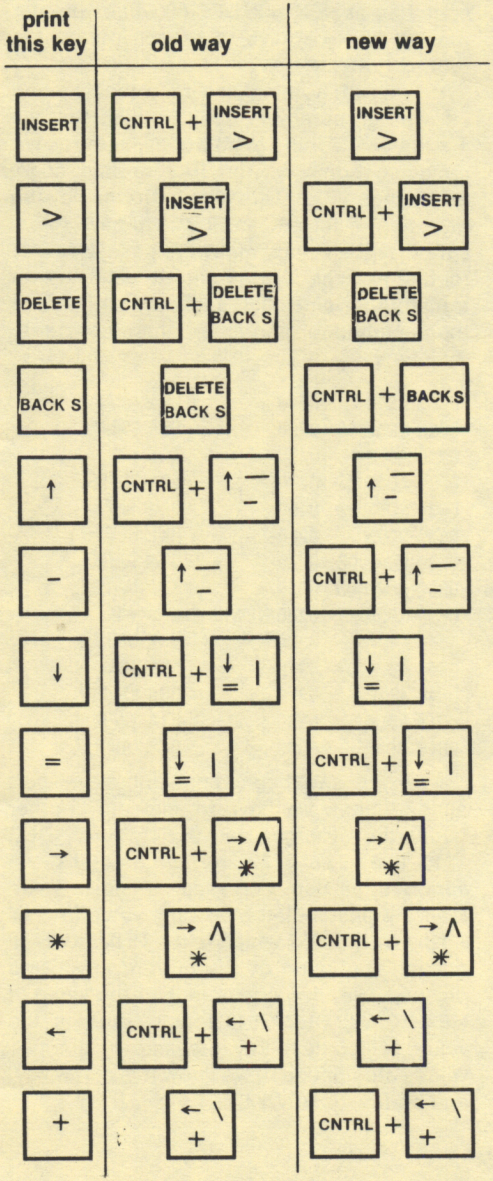
There is also a sound demo program which plays the theme songs from 2001: A Space Odyssey, and Star Wars.

# UTITLITY OF THE MONTH —

**programmer's keyboard**

Our utility of the month is a machine language program that enhances the screen editor on your ATARI. To use it, load in the program, and run it. All you'll see is our copyright message, and then the READY prompt. No, this is not a bug! Here's how it works:

The keyboard is changed as follows:




CNTRL = CONTROL

If you're used to the old keyboard, the new keyboard takes a little getting used to, but it makes line editing a lot easier once you do. There's one more option that will come in handy. If you wish to return to the normal keyboard, press the [OPTION] key, and [CONTROL] + [5] keys all at the same time, and the keyboard will return to its normal state. Press the above keys again, and the keyboard will be re-activated.

The program works by manipulating the keyboard interrupt: whenever a key is pressed, your ATARI stops what it's doing for a split second to figure out which key was pressed, and prints it in the current screen location. The process is called an interrupt. There are three types of interrupts. The first type is a 'chip reset', which occurs only on power up. The second type of interrupt is a Non Maskable Interrupt (NMI). When you press the [SYSTEM RESET] key, you set an NMI into action. You've probably heard of a Display List Interrupt (DLI). This is also an NMI.

The third type of interrupt is a maskable interrupt (IRQ). IRQ stands for an Interrupt Request, and means just that. A request is made to interrupt whatever is currently being done, to see if a key has been pressed. However, the interrupt is only a request. If an NMI is underway, the request is rejected.

The folks at ATARI saw the need to allow a programmer to change the interrupt request routine to do something else, or to add additional duties. Before an interrupt request is executed, decimal locations 208 and 209 are checked to see where the address of the interrupt routine begins. If we want to change IRQ interrupt duties, all we have to do is change the address in 208 and 209 to the location where our routine is.

Thus there are two steps in changing the IRQ interrupt routine. First we must place our machine language routine in a safe area of memory. Then we have to change the pointer in 208 and 209 to point to our routine. Let's take a look at the program:

```
                        *= $0600
0600 AD1F D0  0010      LDA $D01F
0603 C903     0020      CMP #$03
0605 D012     0030      BNE NOOPTION
```

Here we specify where we want to place the machine language routine ($0600 is decimal 1536). Then we load the accumulator with the value located at HEX $D01F, DEC 53279. This is the address that reads the [OPTION], [SELECT], and [START] keys. If the value in this location is equal to 3, the [OPTION] key has not been pressed, and we branch to another part of the routine.

```
0607 ADAF 06  0040          LDA $06AF
060A C900     0050          CMP #$00
060C D008     0060          BNE RESTART
060E A901     0070          LDA #$01
0610 8DAF 06  0080          STA $06AF
0620 4CBE FF  0090          JMP $FFBE
```

If the [OPTION] key has been pressed, we have to find out if the last time the [OPTION] key was pressed the new keyboard routine was active or inactive. If it was active we want to deactivate it. We accomplish this by checking an address we 'POKE'd with a 1 or a 0, depending if it is presently active (=0) or inactive (=1). The temporary storage address we selected is HEX $06AF, DEC 1711. If the [OPTION] key has been pressed, we load the accumulator with the value in this address. If the routine was inactive (=0), we branch to RESTART to activate the keyboard. If it was active we store a 1 in $06AF, and jump to the next section of the interrupt routine in ROM ($FFBE).

```
0616 CEAF 06  0100 RESTART  DEC $06AF
0619 ADAF 06  0110 NOOPTION LDA $06AF
061C C901     0120          CMP #$01
061E D003     0130          BNE SUBROUT
0620 4CBE FF  0140          JMP $FFBE
```

We branch to RESTART if the [OPTION] key has just been pressed, and the keyboard routine was inactive. In this case, we change the value in $06AF from a 1 to a 0, by decrementing the value by 1, and continue the 'new keyboard' processing.

We branch to NOOPTION if the [OPTION] key was not being pressed. Here we might have to check the value in location $06AF again to see if the 'new keyboard' in active. If it is, we branch to the beginning of the subroutine that changes the keys. If it's inactive, we jump out of the routine to the next section of the IRQ routine in ROM.

```
0623 A542     0150 SUBROUT  LDA $42
0625 D074     0160          BNE EXIT
0627 8A       0170          TXA
0628 48       0180          PHA
0629 98       0190          TYA
062A 48       0200          PHA
```

Finally, we can start accomplishing our goal . . . almost. First, we have to check to make sure that critical code is not being executed (i.e. [SYSTEM RESET] pressed, or program loading or saving). If it is, the critical code flag at HEX $42, DEC 66 will be greater than 0, and we have to branch out of the routine (EXIT).

However, if nothing else important is going on, we can continue. Keep in mind that we are not only adding extra duties, but are also replacing some of the IRQ routine code. Since lines 150-200 are also performed by the first step of the IRQ in ROM, we have to ensure that the values that were passed to this routine exit the routine unfettered. Thus we transfer the values in the X and Y registers to the accumulator and 'push' them down into the stack.

```
062B A900     0210          LDA #$00
062D 8DAE 06  0220          STA $06AE
0630 A8       0230          TAY
0631 AD09 D2  0240          LDA $D209
0634 CDF2 02  0250          CMP $2F2
0637 D006     0260          BNE NEWKEY
0639 8DAE 06  0270          STA $06AE
063C 4C4D 06  0280          JMP NOTCTL
063F C99F     0290 NEWKEY   CMP #$9F
0641 D00A     0300          BNE NOTCTL
0643 ADFF 02  0310          LDA $2FF
0646 49FF     0320          EOR #$FF
0648 8DFF 02  0330          STA $2FF
064B B047     0340          BCS OUT
```

Now we can begin changing keys . . . almost! First, we have to zero out another temporary storage area (HEX $06AE, DEC 1710, lines 0210-0220), to help us test if the same key is being pressed. To do this, we load the accumulator with the value in HEX $D209 DEC 53257 which is the ROM address holding the value of the current key being pressed. We then compare it to the value in HEX $02F2, DEC 754, which contains the value of the last key pressed. If the two values are not equal, we know it is a new key, and branch to NEWKEY.

# THE LEARNING CENTER

**"Perhaps the finest educational software that I have ever had the pleasure of reviewing. It's easy for kids to use and effective in teaching basic concepts and skills.... My kids are learning with it right now!"**

Fred D'Ignazio, Educational Editor—*Compute!*, Associate Editor—*Softside*,
Author of bestseller—*Katie and the Computer*

We hope you've been using your ATARI for more than just games ... it is, and can be, a valuable educational tool for you and your children.

Bruce and Dianne Mitchell realized this potential and designed a series of programs for use in their Small World Preschool & Kindergarten located in Durham, N.C. Presented on TV's *PM Magazine* these unique educational programs will introduce your 3-9 year olds to the era of learning with computers. Using the graphics and sound capabilities of the Atari, each program develops a particular skill and reinforces correct responses with happy faces and music.

Beginning with basic concepts such as colors, shapes and the alphabet, your child will progress to an understanding of counting, arithmetic, and language skills.

Widely acclaimed, classroom designed and tested, these unique educational tools are now available to you ... *for your children.*

| SPECIAL SKILLS | Preschool | K | 1 | 2 |
|---|---|---|---|---|
| * Color For The Non-Reader | • | • | | |
| * Name That Color | | • | • | |
| * Like Shape Identification | • | • | • | |
| * Different Shape Identification | • | • | • | |
| Cave Game | • | • | • | • |
| **MATH AND NUMBER SKILLS** | | | | |
| Count With Me | • | • | • | |
| Number Recognition | • | • | • | • |
| Addition | | • | • | |
| Subtraction | | • | • | |
| Add.—Vertical/Horizontal | | • | • | |
| Sub.—Vertical/Horizontal | | • | • | |
| Advanced Addition/Subtraction | | • | • | • |
| Ones and Tens | | | • | • |
| **LANGUAGE SKILLS** | | | | |
| * Alphabet Recognition | | | • | • |
| * Letter Sequence | • | • | • | |
| * Like Symbol Discrimination | • | • | | |
| * Different Symbol Discrimination | • | • | | |

| Pricing Information | Cass. | Disk |
|---|---|---|
| Single Program | 8.95 | n/a |
| Special Skills | 29.95 | 34.95 |
| Language Skills | 24.95 | 29.95 |
| Math and Number Skills | 44.95 | 49.95 |
| Complete Set | 74.95 | 79.95 |
| Edumate™ Light Pen | 34.95 | |

| | Cass. | Disk |
|---|---|---|
| Pre-School | 46.95 | 51.95 |
| Kindergarten | 69.95 | 74.95 |
| First Grade | 64.95 | 69.95 |
| Second Grade | 26.95 | 31.95 |

Add $2.00 for postage & handling

* Compatible with our Edumate™ Light Pen

NOTE: All software requires 8K cassette/16K disk

**programmer's institute™**

310½ W. Franklin Street, P.O. Box 3470, Chapel Hill, NC 27514

**1-800-334-SOFT**

If the same key is being pressed, we don't have to check for a CONTROL-1 key combination (lines 290-340). The [CONTROL] and [1] key combination alternately restarts and stops the screen from scrolling. This is another duty that is performed by the routine we are replacing, so to ensure proper functioning, we include it in our routine.

```
064D C9 07    0350 NOTCTL  CMP #$07;'*' key?
064F F0 4C    0360         BEQ ADD
0651 C9 06    0370         CMP #$06;'+' key?
0653 F0 48    0380         BEQ ADD
0655 C9 0F    0390         CMP #$0F;'=' key?
0657 F0 44    0400         BEQ ADD
0659 C9 0E    0410         CMP #$0E;'-' key?
065B F0 40    0420         BEQ ADD
065D C9 37    0430         CMP #$37;' ' key?
065F F0 3C    0440         BEQ ADD
0661 C9 34    0450         CMP #$34;BACKS'key?
0663 F0 38    0460         BEQ ADD
0665 C9 B4    0470         CMP #$B4;CONTROL+
0667 F0 3A    0480         BEQ SUB
0669 C9 87    0490         CMP #$87; > ?
066B F0 36    0500         BEQ SUB
066D C9 86    0510         CMP #$86;     key?
066F F0 32    0520         BEQ SUB
0671 C9 8F    0530         CMP #$8F;     key?
0673 F0 2E    0540         BEQ SUB
0675 C9 8E    0550         CMP #$8E;     key?
0677 F0 2A    0560         BEQ SUB
0679 C9 BF    0570         CMP #$B7;CONTROL+
067B F0 26    0580         BEQ SUB
```

If the CONTROL-1 keys are not pressed, and all other constraints are satisfied, we can start changing key values. The last value we loaded into the accumulator was from address HEX $D209, DEC 53257 (current key pressed), so we compare the internal key codes for the keys we want to change to the value in the accumulator. If we want to change a non-control key to its control equivalent, we must add HEX $80, DEC 128 to the current key value. To change a control character to a normal value we subtract $80.

```
067D AD09 02  0590         LDA $D209
0680 8DFC 02  0600 CHANGE  STA $2FC
0683 8DF2 02  0610         STA $2F2
0686 ADAE 06  0620         LDA $06AE
0689 C900     0630         CMP #$00
068B D007     0640         BNE OUT
068D A906     0650         LDA #$06
068F 854D     0660         STA $4D
0691 8DF1 02  0670         STA $2F1
```

If any of the specified keys have not been pressed, we reload the accumulator with the current key ($D209) and continue. If one has been pressed, we store the new value in the OS Shadow register of current key ($2FC shadow for $D209), and store the key in the shadow address for the last key pressed ($2F2).

We've finally almost finished our routine. But first, we have to check our temporary storage location ($06AE) to see if this key was the same as the last key pressed. (Remember that a 1 in this address means it is the same key, a 0 means it's different). If it is the same we branch to OUT. Why? Well, locations HEX $4D and $2F1, DEC 77 and 753 respectively, are the shadow registers for ATTRACT and KEYDEL. If you don't press any keys for about 9 minutes, your screen starts to change colors. When a key is pressed, we should restore its normal value (0, or one close to it) to this register. The reason why we don't have to store a 0 in ATTRACT is that the POKEY timers add 1 to this address every 3 seconds or so. When the value in ATTRACT is greater than 128, the colors start to change. If we put a 3 or 4 in ATTRACT, all we're doing is subtracting a couple of seconds from the 9 minute normal screen.

KEYDEL is the debounce delay timer. It is set to 3 whenever a key is pressed and decremented by 1 every 60th of a second. Thus, if a key is pressed, we must also update this register.

```
0694 8C2B 02  0680 OUT     STY $22B
0697 68       0690         PLA
0698 A8       0700         TAY
0699 68       0710         PLA
069A A8       0720         TAX
069B 68       0730 EXIT    PLA
069C 40       0740         RTI
069D 18       0750 ADD     CLC
069E 6980     0760         ADC #$80
06A0 4C80 06  0770         JMP CHANGE
06A3 38       0780 SUB     SEC
06A4 E980     0790         SBC #$80
06A6 4C80 06  0800         JMP CHANGE
```

Before we can exit the routine, we have to eliminate the keyboard repeat function, because it is done in the second section of the IRQ routine, which is lengthy and inefficient to copy. If we don't stop the keys from

repeating, the keys we changed will return to their normal value once it starts to repeat. We stop the repeat function by storing a 0 in HEX $22B, DEC 555. This location is called SRTIMR, and is used to time the difference between when a key is pressed and when it starts to repeat. A value of 0 cancels this process.

Then we restore the original values in the X and Y registers by pulling back off the stack the values we pushed into it earlier in the routine. We return from the interrupt, go on to another part of the IRQ routine, and our goal is accomplished!

## 6502 TUTORIAL - #1

Our machine language tutorials will combine text in our magazine, and examples from programs on cassette or diskette. Before reading further, load and run the **6502 Tutorial Program.**

When you first bought your ATARI, you probably thought you'd never truly understand ATARI BASIC. Months later, armed with the decision that you've mastered BASIC, you begin to notice the limitations of the language . . . after all, you can only do one thing at a time, and it accomplishes this while working at a snail's pace.

But let's not cut BASIC to pieces. In many ways it performs admirably . . . if only you could speed up a couple subroutines, or maybe play some music while your program is slaving away. Well, you can do these things and more, but you have to advance to the next step. You have to learn 6502 Assembly Language Programming.

Where should we begin? We could explain all the 6502 Operands (like a BASIC Command), but that would most definitely put everyone to sleep! A much better way would be to show you some simple machine language programs and subroutines, and explain the 6502 'commands' as they appear. So this is what we'll do.

However, knowing the 6502 operands alone won't make you a 6502 whiz. You have to understand what makes up the 6502 chip, how it works; and its working relationship with the other chips in the ATARI (ANTIC, POKEY, GTIA, etc.), and the Operating System.

It's really very simple. All you have to do is keep in mind that your computer does everything you tell it to by turning bits on and off. What is a bit? Well, a 16K ATARI contains 16K bytes (also called addresses). A byte can hold any number (decimal) from 0 to 255, and is made up of 8 bits. By "ON" or "OFF" we mean that a bit will contain either a 1 (ON) or a 0 (OFF). Complete the first procedure in our **6502 Tutorial Program #1.**

Now that you understand all about binary numbers, and bits and bytes, let's talk about the 6502 chip.

The 6502 is an 8 bit chip. By this we mean that signals are sent to the 6502 8 bits at a time. The 6502 is one of the easiest chips to understand. All the work is done by the following:

**An 8-bit ACCUMULATOR.** The accumulator is used as temporary storage, and is the most used register in the 6502.

**An 8-bit X-REGISTER.** The X-Register is also used for temporary storage, but can also be incremented and decremented by a programmed instruction.

**The Y-REGISTER** performs the same function as the X-Register. These two registers can thus be used to examine successive locations in memory. In other words, you can use these 2 registers to look at the contents of one address, and then add or subtract one from the registers, and look at the contents of the next higher or lower memory address. Thus they can also be used in a similar way as FOR . . . NEXT loops in BASIC.

**a 16-bit PROGRAM COUNTER.** The program counter is used to make sure that instructions in a program are executed in the proper order.

**A 16-bit STACK.** The Stack is used to store data by a last in - first out method. In other words, you 'push' data down into the stack, and 'pop' them out when you wish to retrieve them.

The last element of the 6502 hardware is the **8-bit PROCESSOR STATUS REGISTER.** Each bit in the Processor Status Register 'flags' (i.e. identifies) a certain condition:

| Bit no. | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Flag | N | V | — | B | D | I | Z | C |

Bit 0 is the carry bit. If an operation is performed, and the result is greater than 255, the CARRY BIT is SET to 1 (it is normally 0) to show that the result is greater than a 1 byte address can hold.

Bit 1 is the ZERO bit. If an operation is performed, and the result of the operation is 0, this bit is set to 1 (it is normally zero).

The only other bit we will explain at this point is Bit 7. When an operation is performed, and the result is less than 0, the N bit will be set to 1 (it is normally 0).

Okay, we understand what the 6502 is made up of. Now let's complete the next procedure in the 6502 Tutorial Program — adding 2 numbers. Notice how we used the accumulator as temporary storage, and the carry bit of the processor status register to check for a byte overflow.

If the result of the addition is greater than 255, trying to POKE this result in address 1602 will give you an ERROR-3. We must guard against this by checking the carry bit of the processor status register to see if it's set. Here is the BASIC equivalent of the assembly language addition program:

```
10 X=235: POKE 1600, X
20 Y=50: POKE 1601, Y
30 Z=PEEK (1600) + PEEK (1601)
40 IF Z  255 THEN 70
50 POKE 1602, Z-255
60 POKE 1603, 1: GOTO 90
70 POKE 1602, Z
80 POKE 1603, 0
90 END
```

Here we stored the result in two successive addresses. We'll close this first lesson, with a program that fills the screen with inverse characters. We accomplish this by finding the beginning of screen memory, and creating a loop within a loop that fills the GRAPHICS 0 screen. There are some features of this program that need further comment (The fully commented listing of the program is below. Compare it to the BASIC version in the program).

First we accessed the routine through a USR call from BASIC. The USR command should not be considered mysterious. Its format is:

V = USR (address, V1, V2, etc.)

where V is a value that is returned from the assembly language routine if necessary. This is a rarely used feature that we will explain in a future lesson. The 'address' tells the BASIC interpreter where the assembly language routine starts. V1, V2, etc. are other variables that you might want to pass to the machine language subroutine. This is another feature that we'll explain later.

Second, notice that the first instruction in the subroutine was a PLA (pull one byte of data off the stack and place it in the accumulator). The reason why we must do this first is that the USR call passes at least two variables to the subroutine by pushing them onto the stack before beginning the machine language subroutine. The first variable is the address in the BASIC program where the machine language routine is supposed to return when it's finished. This is actually sent in two bytes (a low byte, and a high byte). The second variable passed is the number of variables passed. In the case presented here, 0 variables are passed, so this value must be pulled off the stack and discarded.

Third, notice how we used a page-0 (locations 0-255 decimal) set of addresses, and the Y-Register to index through screen RAM one additional address at a time. We used an address in page 0 that is unused by the BASIC cartridge and the Operating System. Take note that this form of indexing (one that points to the address contained in two successive bytes) can only be done using a page 0 address.

```
0000                10          *= $0600 ;
0600 68             20          PLA
0601 A2 00          30          LDX #$00
0603 A5 58          40          LDA $58
0605 85 D2          50          STA $D2
0607 A5 59          60          LDA $59
0609 85 D3          70          STA $D3
```

Here we pull the first byte off the stack (number of variables passed which in this case is 0). Then we transfer the low byte and high byte of beginning screen RAM ($58, $59

HEX/88, 89 DEC) to $D2 and $D3 HEX (210, 211 DEC). We do this so we can change the contents of these two bytes without changing the location where screen RAM starts. (In other words, 88 and 89 are used by the Operating System so if we add to the values in this location, we'll still be printing at the first location of screen RAM — we'll just be moving the whole page through memory.

```
060B A9 80     80 LOOP2    LDA #$80
060D A0 00     90          LDY #$00
060F 91 D2    100 LOOP1    STA ($D2),Y
0611 C8       110          INY
0612 C0 F0    120          CPY #$F0
0614 D0 F9    130          BNE LOOP1
```

Now we start our big loop by loading the accumulator with the character we want to print ($80 HEX is the internal code for an inverse space). We use the Y register as an index that starts at 0 and increments to 240 4 times (4 X 240 = 960 bytes — the number of bytes in a GRAPHICS 0 screen). We take the value in the accumulator and store it in screen RAM starting at the first byte up to the 240th byte. Notice how we increment the value in the Y Register by one each time through LOOP1 with the INY command. When Y reaches 240 we do not branch back to LOOP1, but move further down the routine.

```
0616 18       140          CLC
0617 98       150          TYA
0618 65 D2    160          ADC $D2
061A 85 D2    170          STA $D2
061C 90 02    180          BCC NOCARRY
061E E6 D3    190          INC $D3
0620 E8       200 NOCARRY  INX
0621 E0 04    210          CPX #$04
0623 D0 E6    220          BNE LOOP2
0625 60       230          RTS
0626          240          .END
```

This part of the routine should look very familiar . . . because it is! All we're doing in lines 140 to 200 is tranferring the value in the Y register (240 DEC after each LOOP1 is complete) to the accumulator and adding it to the value in 210 and 211. We then increase the value in the X Register by 1 and compare it to 4. If it's less than 4, we branch back to the beginning of LOOP2, and continue storing inverse spaces in screen RAM, but starting 240 bytes more after the beginning of screen RAM each time through the loop. When the value in the X-Register is equal to 4, we return from the machine language subroutine, and the task is complete.

In the next issue, we'll show you some more machine language routines, and the relationships between the OS, the BASIC Cartridge, and pages 0-5. Until then, please write and let us know things you want to see done in assembly language, and what you do and don't understand.

# —New Dealers and Distributors— Of Our Products

Video Etc.
465 Lake Cook Rd.
Deerfield, IL 60015

Software City
106 El Paseo Center
San Jose, CA 95130

Madison Books & Computers
8006-5 Madison Ave.
Madison, AL 35758

Micro Solutions Inc.
9949 Harwin E.
Houston, TX 77036

Micro Innovations
504 Ridge Ave.
St. Louis, MO 63119

Teacher's Corner
1338 10th St.
Gering, NE 69341

3E Software & Systems
22408 Mission Blvd.
Haywood, CA 94541

Micro-Age Computer Store
1707 Monroe Ave.
Rochester, NY 14618

The Software Connection
5460 N. St. RD 7
Suite 108
Ft. Lauderdale, FL 33319

Leigh's Computers
212 E. 85th St.
New York, NY 10028

Bit Bucket
1355 Washing St.
W. Newton, MA 02165

Level IV Products, Inc.
32429 Schoolcraft Rd.
Livonia, MI 48150

The Computer Terminal
908 Bacons Bridge Rd.
Summerville SC 29483

Computer Mania
154 E. Little Creek Rd.
Norfolk, VA 23505

Rainbow Computer Systems
1085 Helena Ave.
Helena, MT 59601

CMD Micro
10546 106 St.
Edmonton, Alberta
Canada T5H2X6

Softcel
26 Great Corn Bow
Halesowen, W. Midlands
England

Greg Wilson
800½ Macleary St.
Potts Point 2011
Australia

# programmer's institute

P.O. Box 3470
Chapel Hill, North Carolina 27514
(919) 967-0861

Name _____ Phone (days) _____

Address _____

City _____ State ____ Zip _____

| Computer | Name of Product | Quantity | Price Ea. | Total |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

**Prices are subject to change without notice.**

| | |
|---|---|
| Sub-Total | |
| Postage & Handling | $2.00 |
| Total Enclosed | |

☐ Check or Money Order enclosed
☐ MasterCard
☐ VISA

Credit Card No. _____

Expiration Date _____

Full-year overseas subscriptions add $25.00

HOURS: Mon - Fri 10-6, Sat 11-3, Technical Questions 6-7 pm

## order direct: 1-800-334-SOFT